Mohamed Manessouri  
SSE  
  
 **HOW TO CODE AND IMPLEMENT DESIGN IN THE FPGA CARD**

**1. Write the VHDL Code and Check Syntax**

* Writing the VHDL code and checking the syntax is a fundamental step. Simulation (optional) can be done here using test benches or manual value forcing.

**2. Write a UCF File or Configure I/O Pins**

* This step involves defining how your logical signals in the VHDL design map to the physical pins of the FPGA.

This can be done in two ways: Coding or configuring it manually in I/O Pin Planning (PlanAhead) –Pre-Synthesis.

**3. Synthesize - XST**

* This step translates your VHDL code into a netlist, which is a hardware representation of your design.

**4. Implement Design**

* The steps of **Translate**, **Map**, and **Place & Route** collectively prepare your design for implementation on the FPGA:
  + **Translate:** Checks for design rule violations and combines design files.
  + **Map:** Assigns logic to specific FPGA resources.
  + **Place & Route:** Allocates the mapped resources to specific physical locations and connects them.

**5. Generate Programming File**

* Converts the placed and routed design into a bitstream file that the FPGA can understand.

**Plug the card with your pc and alimentation:**

**6. Configure the FPGA**

* Using tools like **iMPACT**, load the bitstream file to the FPGA. The **PROM/ACE file generation** is indeed optional and used primarily for persistent configuration.

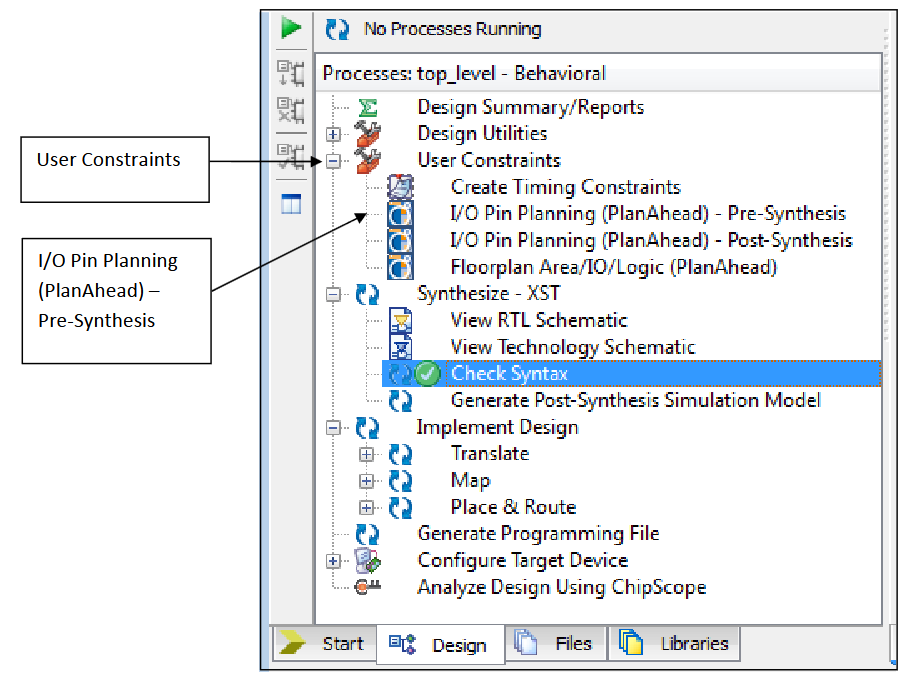
**7. Test the Design**

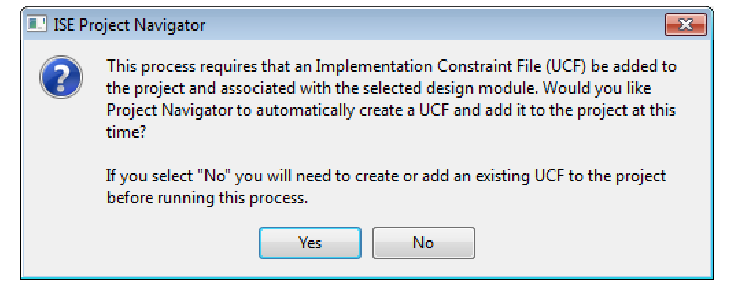
* After successful programming, verify that your design behaves as expected on the hardware.

**1. Write the VHDL Code and Check Syntax**

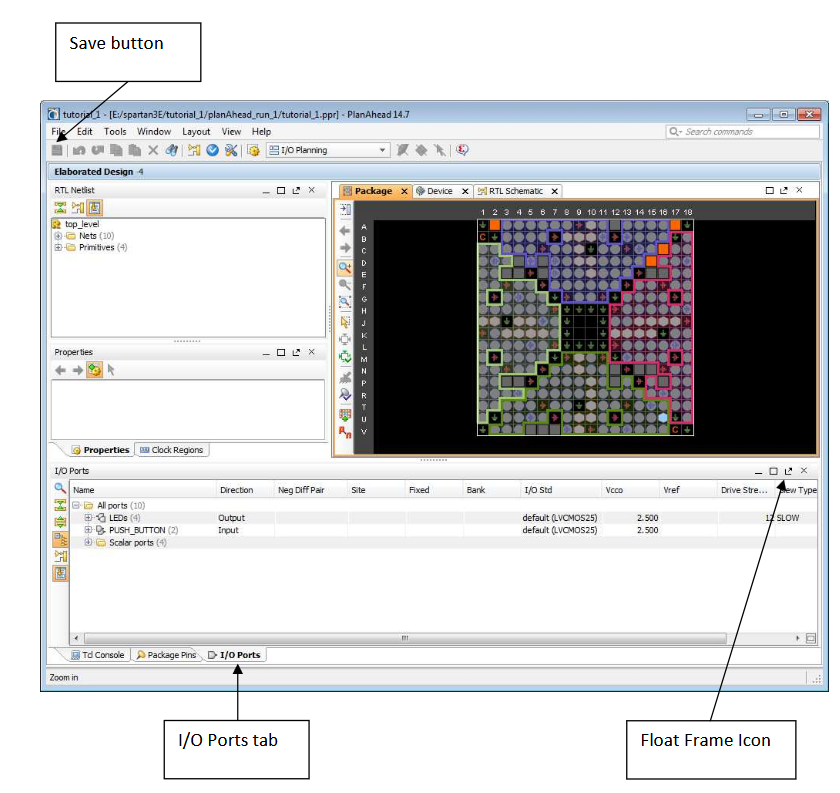
This is clear I think. Writing your vhdl code and check if there is any error   
  
Simulation is required using test benches or forcing values to understand the behavior.   
  
  
we are going to focus now on the other steps:

**2. Write a UCF File or Configure I/O Pins**  
configuring it manually in I/O Pin Planning (PlanAhead) – Pre-Synthesis:

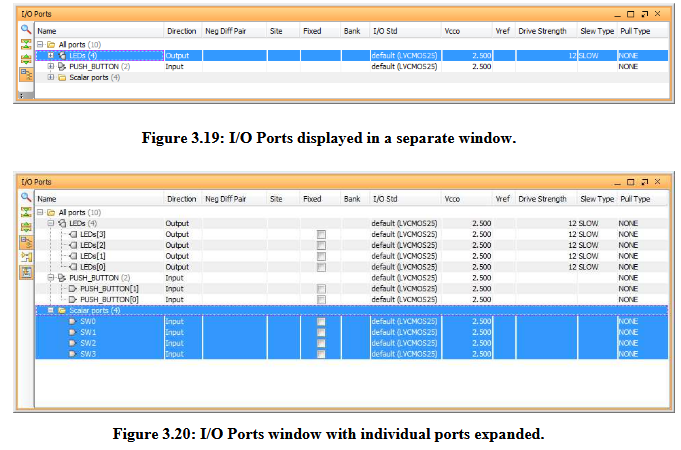
1)   
  


2) 

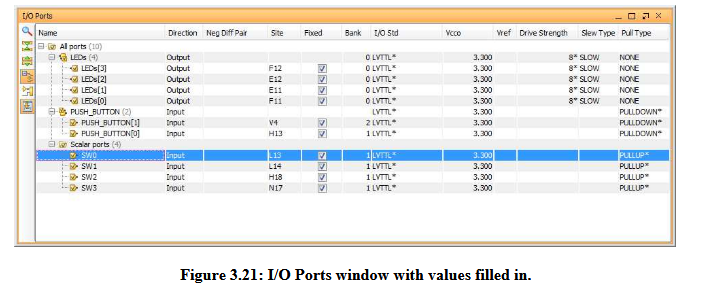
3)



4)



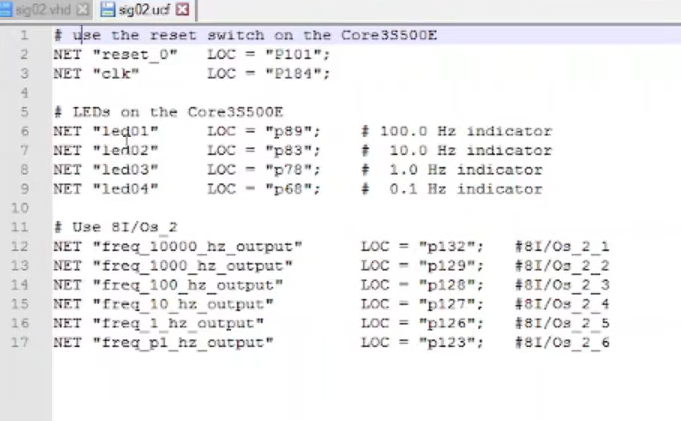
5)



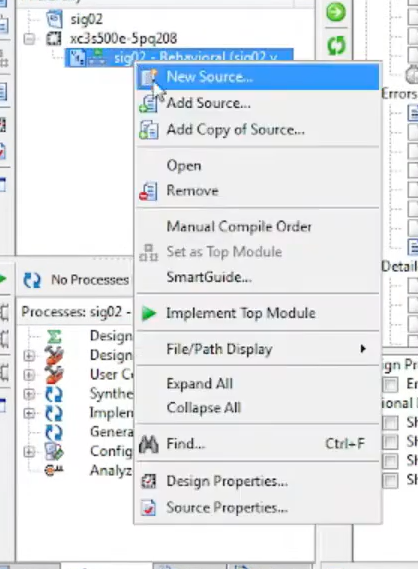
6) click save and a ucf file would be generated

configuring it directly by writing a ucf file:

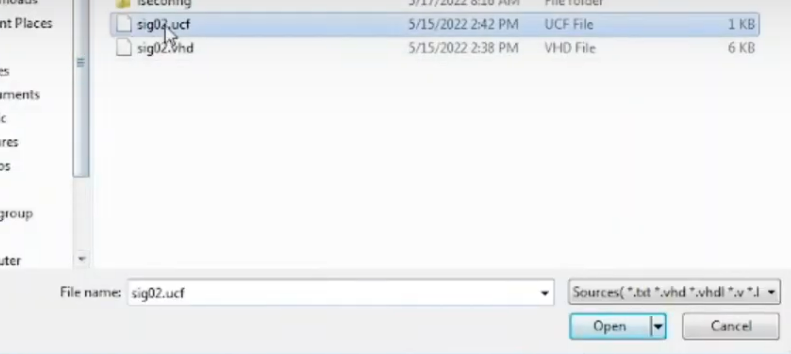
* 1. Example





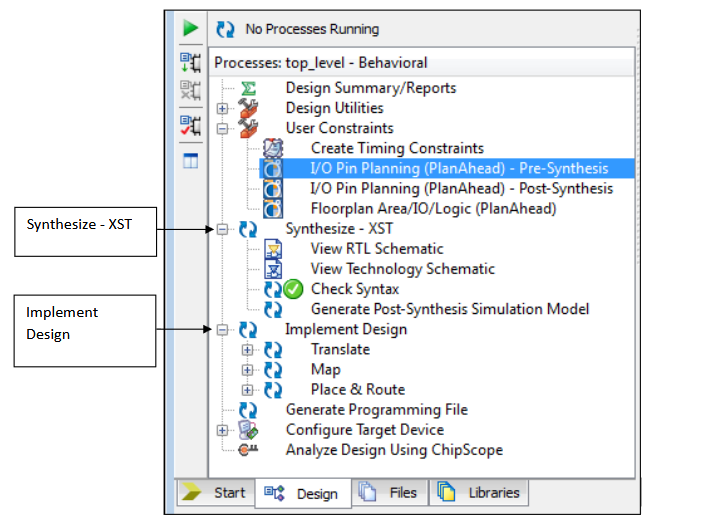
****



****

**3. Synthesize - XST**

**1)**

****

### 2)

### ****What is Synthesis?****

Synthesis is the process of converting your VHDL or Verilog code (a high-level description of your design) into a **netlist**, which is a lower-level hardware representation.

### ****3)****

### ****What is XST?****

XST stands for **Xilinx Synthesis Technology**, and it’s the tool within Xilinx ISE that performs the synthesis process. It reads your VHDL code, analyzes it, and translates it into a structure made of basic digital components like flip-flops, LUTs (Look-Up Tables), multiplexers, etc., which are then mapped to the physical hardware of the FPGA.

### 4)

**Workflow with These Tools**

1. **Before Running Synthesis:**
   * Use **Check Syntax** to validate your VHDL code.
2. **After Running Synthesis:**
   * Use **View RTL Schematics** to verify the logical structure of your design.
   * Use **View Technology Schematics** to inspect how your design maps to hardware.
   * If needed, **Generate Post-Synthesis Simulation Model** and run simulations for further validation.

**4. Implement Design**

### This is where your design gets prepared for the FPGA hardware after synthesis. It consists of **three sub-steps**: Translate, Map, and Place & Route.

**1. Translate**

* The **Translate** step combines all the inputs (e.g., the synthesized netlist, UCF constraints, etc.) into a single file that the FPGA tools can work with.

**What it does:**

1. Checks for design rule violations (e.g., missing pin assignments, invalid constraints).
2. Integrates your netlist and constraints to create a complete design representation.

**Outputs:**

* A file ready for mapping, often in a format specific to the FPGA tool (e.g., NGC or NGD format in Xilinx ISE).

**Purpose:**

* Ensures that your design files are consistent and complete before proceeding.

**2. Map**

**What it is:**

* The **Map** step assigns the logic elements from the netlist (like LUTs, flip-flops, and RAMs) to the specific resources available in your FPGA.

**What it does:**

1. Matches logical components to FPGA resources.
   * Example: An AND gate in your design will be assigned to a specific part of a LUT.
2. Checks for resource conflicts (e.g., if too many components are assigned to the same area).

**Outputs:**

* A mapped design file that shows how your logic maps to FPGA resources.

**Purpose:**

* Ensures that your design can fit into the FPGA and efficiently utilizes its resources.

**3. Place & Route (P&R)**

**What it is:**

* This is where the mapped logic is physically placed on the FPGA, and the connections (routes) between components are made.

**What it does:**

1. **Placement:** Determines where each logic block will physically reside on the FPGA.
   * Example: LUTs, flip-flops, and other components are assigned to specific locations.
2. **Routing:** Establishes the physical paths for signals to travel between components.

**Challenges Addressed:**

* Timing: Ensures signals meet timing requirements (e.g., clock signals arrive on time).
* Congestion: Avoids overcrowding specific areas of the FPGA.

**Outputs:**

* A fully placed and routed design file.

**Purpose:**

* Ensures that the design meets both functional and timing requirements when implemented on the FPGA.

### The result of these steps is an intermediate file (like an **NGD file**) that represents your design as it fits into the FPGA.

### during the **Implement Design** phase, the FPGA card does **not need to be plugged into your PC**. This phase is purely a **software process** that prepares your design for the hardware.

**5. Generate Programming File**

The **Generate Programming File** step is where you create the final file that will be loaded onto your FPGA. This file configures the FPGA to implement your design.

**What is a Programming File?**

* The programming file (usually with a .bit extension) is a binary file that contains the **configuration data** for the FPGA.
* This file is generated after the **Translate, Map, and Place & Route** steps are successfully completed.

 **Implement Design:** Prepares the design (logical mapping, routing, etc.).

###  **Generate Programming File:** Converts the prepared design into the final .bit file for the FPGA.

**Requirements for This Step**

1. **Hardware Setup:**
   * Connect your FPGA development board (e.g., Spartan 3E) to your PC using a compatible cable, such as a JTAG or USB programmer.
   * Power up your FPGA board.
2. **Software Setup:**
   * Use Xilinx **iMPACT** (part of Xilinx ISE) to perform the configuration.
   * Ensure the correct bitstream file (.bit) has been generated.

**6. Configure the FPGA**

### 